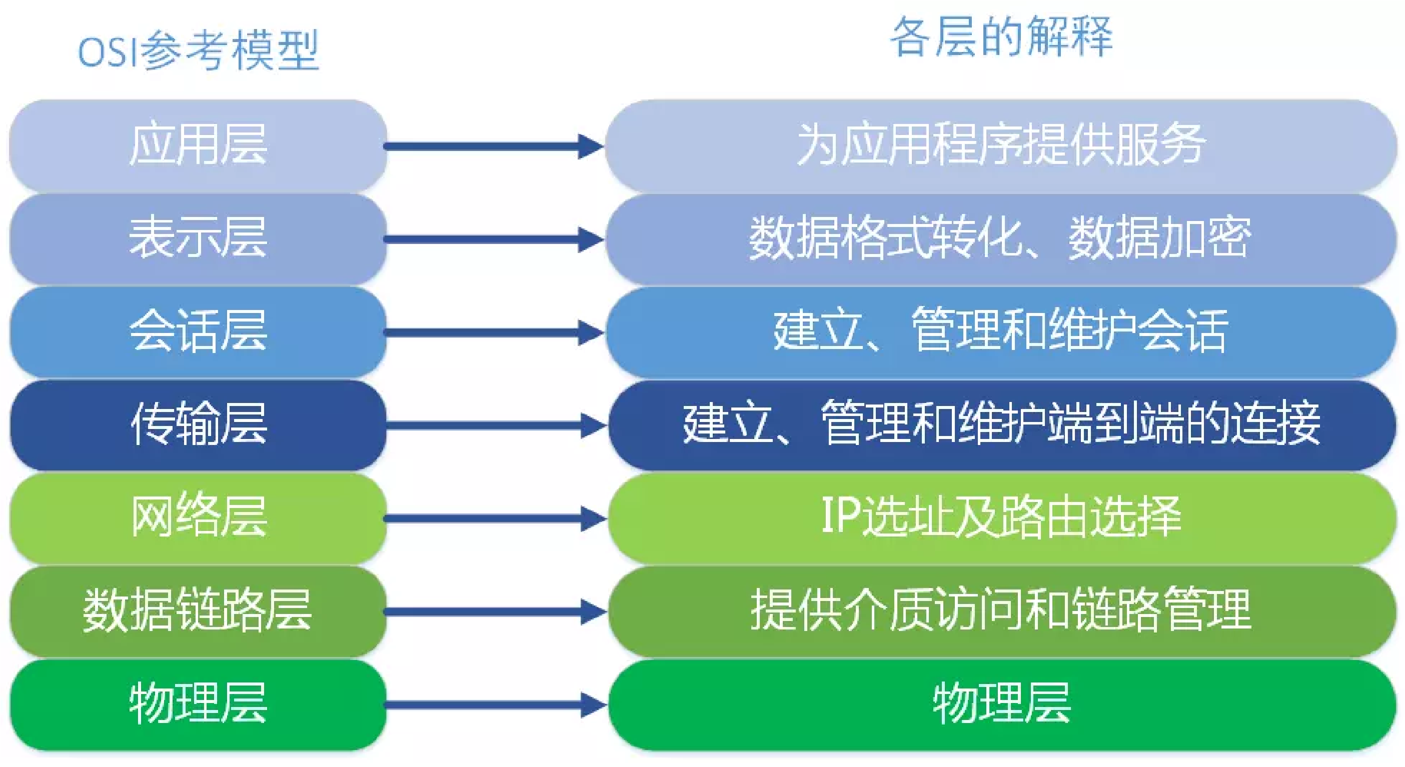
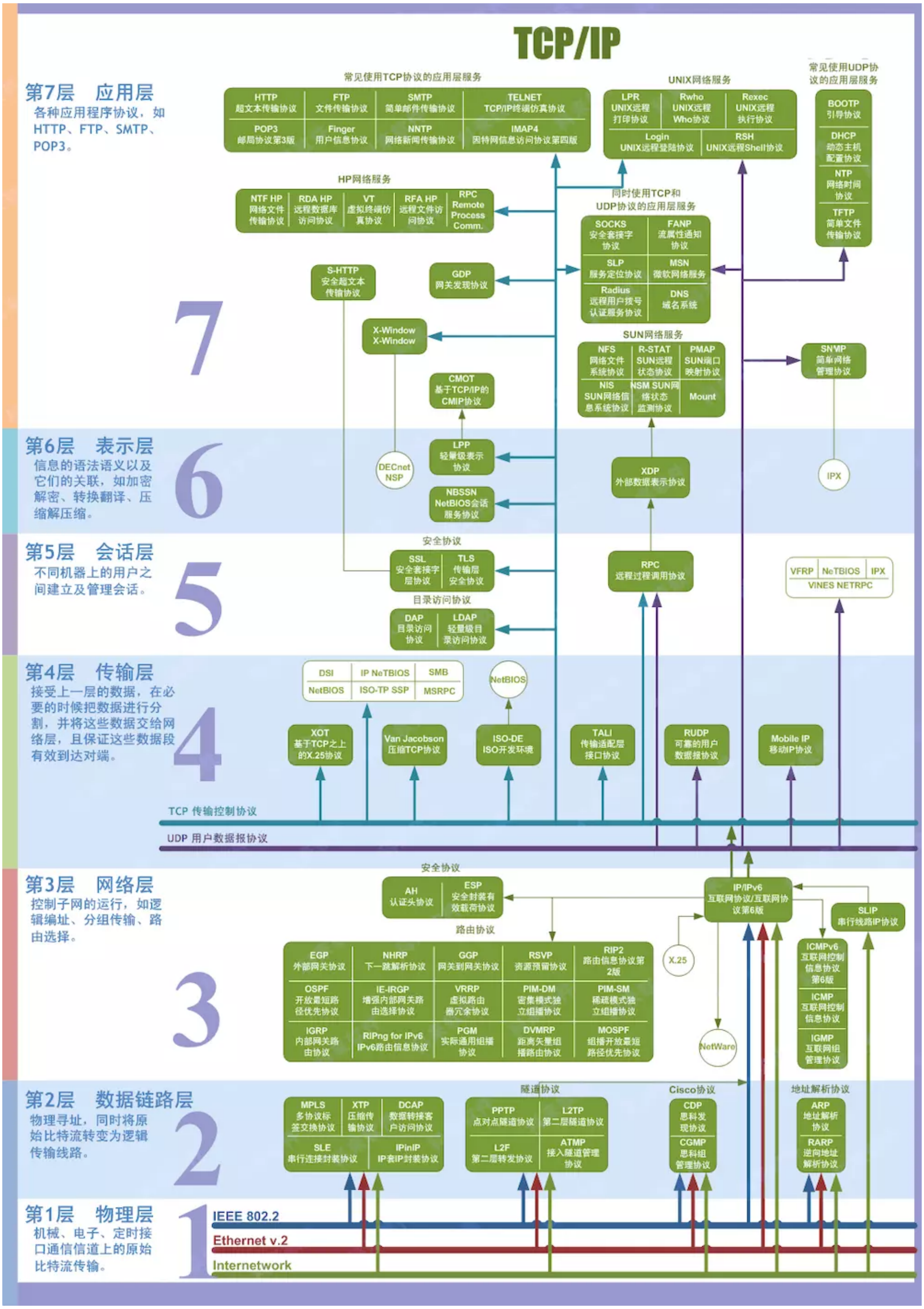
NET



OSI参考模型	各层的解释
应用层	为应用程序提供服务
表示层	数据格式转化、数据加密
会话层	建立、管理和维护会话
传输层	建立、管理和维护端到端的连接
网络层	IP选址及路由选择
数据链路层	提供介质访问和链路管理
物理层	物理层


TCP/IP
第7层 应用层
各种应用程序协议，如HTTP、FTP、SMTP、POP3。
7
常见使用TCP协议的应用层服务
HTTP 超文本传输协议
FTP 文件传输协议
SMTP 简单邮件传输协议
TELNET TCP/IP终端仿真协议
POP3 邮局协议第3版
Finger 用户信息协议
NNTP 网络新闻传输协议
IMAP4 因特网信息访问协议第四版
UNIX网络服务
LPR UNIX远程打印协议
Rwho UNIX远程Who协议
Rexec UNIX远程执行协议
Login UNIX远程登陆协议
RSH UNIX远程Shell协议
常见使用UDP协议的应用层服务
BOOTP 引导协议
DHCP 动态主机配置协议
NTP 网络时间协议
TFTP 简单文件传输协议
HP网络服务
NTF HP 网络文件传输协议
RDA HP 远程数据库访问协议
VT 虚拟终端仿真协议
RFA HP 远程文件访问协议
RPC Remote Process Comm.
同时使用TCP和UDP协议的应用层服务
SOCKS 安全套接字协议
FANP 流属性通知协议
SLP 服务定位协议
MSN 微软网络服务
Radius 远程用户拨号认证服务协议
DNS 域名系统
S-HTTP 安全超文本传输协议
GDP 网关发现协议
X-Window X-Window
SUN网络服务
NFS 网络文件系统协议
R-STAT SUN远程状态协议
PMAP SUN端口映射协议
NIS SUN网络信息系统协议
NSM SUN网络状态监测协议
Mount
SNMP 简单网络管理协议
CMOT 基于TCP/IP的CMIP协议
第6层 表示层
信息的语法语义以及它们的关联，如加密解密、转换翻译、压缩解压缩。
6
LPP 轻量级表示协议
DECnet NSP
NBSSN NetBIOS会话服务协议
XDP 外部数据表示协议
IPX
第5层 会话层
不同机器上的用户之间建立及管理会话。
5
安全协议
SSL 安全套接字层协议
TLS 传输层安全协议
RPC 远程过程调用协议
VFRP
NeTBIOS
IPX
VINES NETRPC
目录访问协议
DAP 目录访问协议
LDAP 轻量级目录访问协议
第4层 传输层
接受上一层的数据，在必要的时候把数据进行分割，并将这些数据交给网络层，且保证这些数据段有效到达对端。
4
DSI
IP NeTBIOS
SMB
NetBIOS
ISO-TP SSP
MSRPC
NetBIOS
XOT 基于TCP之上的X.25协议
Van Jacobson 压缩TCP协议
ISO-DE ISO开发环境
TALI 传输适配层接口协议
RUDP 可靠的用户数据报协议
Mobile IP 移动IP协议
TCP 传输控制协议
UDP 用户数据报协议
第3层 网络层
控制子网的运行，如逻辑编址、分组传输、路由选择。
3
安全协议
AH 认证头协议
ESP 安全封装有效载荷协议
IP/IPv6 互联网协议/互联网协议第6版
SLIP 串行线路IP协议
路由协议
EGP 外部网关协议
NHRP 下一跳解析协议
GGP 网关到网关协议
RSVP 资源预留协议
RIP2 路由信息协议第2版
OSPF 开放最短路径优先协议
IE-IRGP 增强内部网关路由选择协议
VRRP 虚拟路由器冗余协议
PIM-DM 密集模式独立组播协议
PIM-SM 稀疏模式独立组播协议
IGRP 内部网关路由协议
RIPng for IPv6 IPv6路由信息协议
PGM 实际通用组播协议
DVMRP 距离矢量组播路由协议
MOSPF 组播开放最短路径优先协议
X.25
NetWare
ICMPv6 互联网控制信息协议第6版
ICMP 互联网控制信息协议
IGMP 互联网组管理协议
第2层 数据链路层
物理寻址，同时将原始比特流转变为逻辑传输线路。
2
MPLS 多协议标签交换协议
XTP 压缩传输协议
DCAP 数据转接客户访问协议
SLE 串行连接封装协议
IPinIP IP套IP封装协议
隧道协议
PPTP 点对点隧道协议
L2TP 第二层隧道协议
L2F 第二层转发协议
ATMP 接入隧道管理协议
Cisco协议
CDP 思科发现协议
CGMP 思科组管理协议
地址解析协议
ARP 地址解析协议
RARP 逆向地址解析协议
第1层 物理层
机械、电子、定时接口通信信道上的原始比特流传输。
1
IEEE 802.2
Ethernet v.2
Internetwork

net dgram http https

Node

net	TCP
dgram	UDP
http	HTTP
https	HTTPS

- TCP 7
- UDP

- IP IP IP
- IP
- IP ICMP
- ARP IP MAC MAC
- IP

0 1

url

url

3

- – SYN – –
- – SYN/ACK – –
- – ACK – –

4

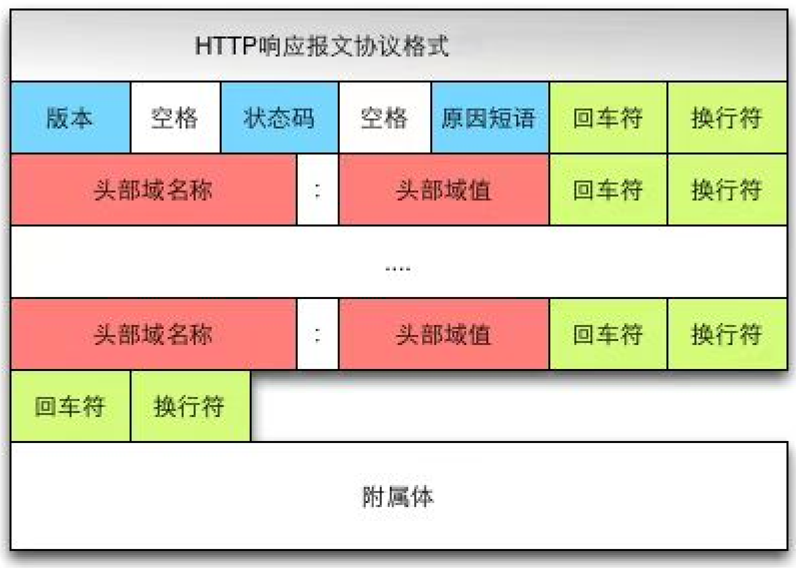
- - FIN
- - FIN ACK 1 SYN FIN
- - FIN
- - ACK 1

```
   IP
 DNS          ->  ->   ...
web      HTTP
     http://example.com  http://www.example.com



  HTTP
 HTML
     HTML          CSS JS
```

1XX	2XX	3XX	4XX	5XX
	200 OK	301	400	500
	204	302 ( )	401 HTTP	503 307 302
	206	303 URL GET	403	
		304 GET	404	

http/https

`http` `TCP/IP` `https` `TLS SSL` `http/https` `TCP` `http` `http`



```
HTTP/1.0 200 OK    //


Content-type:text/plain    //
Content-length:19            //


Hi I'm a message!    //
```

Node `http` net http

API

TCP UDP

Node net TCP API

`net`

```
var net = require("net")
```

`net.createServer([options][, connectionListener])`	TCP connectionListener 'connection'
`net.connect(options[, connectionListener])`	'net.Socket' socket 'connect'
`net.createConnection(options[, connectionListener])`	port host TCP host 'localhost'
`net.connect(port[, host][, connectListener])`	port host TCP host 'localhost' connec 'connect' 'net.Socket'

`net.createConnection(port[, host][, connectListener])`	port host TCP host 'localhost''connect'connec 'net.Socket'
`net.connect(path[, connectListener])`	path unix socket connectListener 'conne
`net.createConnection(path[, connectListener])`	path unix socket connectListener 'conne
`net.isIP(input)`	IP IPV4 4 IPV6 6 0
`net.isIPv4(input)`	IPV4 true false
`net.isIPv6(input)`	IPV6 true false

- `net.Socket` TCP UNIX Socket
- `net.createServer`
- `net.Socket` `net.connect`

net.Server

`net.Server` `TCP`

`server.listen(port[, host][, backlog][, callback])`	port host ac host IPv4 (INADDR_AN
`server.listen(path[, callback])`	path socket
`server.listen(handle[, callback])`	
`server.listen(options[, callback])`	options port, host, backlog, callback , [host], [backlog], [callback]) path UNIX socket
`server.close([callback])`	'close'
`server.address()`	
`server.unref()`	unref
`server.ref()`	unref unref ref
`server.getConnections(callback)`	socket 2 err count

```
let server = net.createServer((socket) => {});
server.listen(3000, () => {});
```

net.Socket

`net.Socket` TCP UNIX Socket net.Socket (connect()) , Noc

- listening server.listen
- connection socket net.Socket
- close

`lookup`	UNIX sokcet
`connect`	socket
`data`	
`end`	socket FIN
`timeout`	socket socket
`drain`	
`error`	
`close`	socket had_error socket

```
let server = net.createServer((socket) => {
    socket.on('data', (data) => {});
    socket.on('end', () => {});
    socket.on('error', (err) => {});
    socket.on('close', () => {});
});
server.on('close', (socket) => {});
server.on('error', (e) => {});
```

net.Sockets

`net.Socket` `socket`

`socket.connect(path[, connectListener])`	unix socket net.createConnection socket
`socket.setEncoding([encoding])`	
`socket.write(data[, encoding][, callback])`	socket UTF8
`socket.end([data][, encoding])`	socket FIN
`socket.destroy()`	I/O
`socket.pause()`	data
`socket.resume()`	pause()
`socket.setTimeout(timeout[, callback])`	socket timeout socket
`socket.setNoDelay([noDelay])`	Nagle TCP noDelay tr

`socket.setKeepAlive([enable][, initialDelay])`	/ socket probe false
`socket.address()`	3 { port: 12346, family: 'I
`socket.unref()`	unref unref unref
`socket.ref()`	unref unref ref

`new net.Socket([options])` `socket`

```
let server = net.createServer((socket) => {
    socket.setEncoding('utf8');
    socket.write();
    socket.end();
});
```

TCP HTTP

net http

`socket.pipe` `fs.createWriteStream` `message.txt`

`http://localhost:3000` hello

```
let net = require('net');
let server = net.createServer({
    //   pauseOnConnect     true,
    pauseOnConnect: true
}, (socket) => {
    socket.setEncoding('utf8');
    socket.on('data', (data) => {
        console.log(data);
    });
    socket.on('end', () => {
        console.log('client disconnected');
    });
    //
    socket.on('error', (err) => {
        console.log("error");
    });
    socket.on('close', () => {
        console.log("close socket");
    });
    socket.end(`
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 5

hello`)
    console.log('request');
});

server.listen(3000, () => {
    console.log('opened server on', server.address({}));
});
server.on('connection', (socket) => {
    console.log('connection');
});

//server.unref();// node server
//
server.on('close', (socket) => {
    console.log('close server');
});
server.on('error', (e) => {
    if (e.code === 'EADDRINUSE') {
        console.log('Address in use, retrying...');
        setTimeout(() => {
            server.close();
            server.listen(PORT, HOST);
        }, 1000);
    }
});
```

`postman` `message.txt`

```
POST /abc HTTP/1.1
Content-Type: multipart/form-data; boundary=--------------------------879095998142409176007484
abc: 123
bbb: ccc
ddd: eee
token:

```

```
eyJkYXRhIjp7ImlucHV0RW1haWwiOiJsZW1vbiIsImlucHV0UGFzc3dvcmQiOiIxMjMifSwiY3JlYXRlZCI6MTU0NzA0MTE
cache-control: no-cache
Postman-Token: 97b4950a-1169-407b-8787-ab238d3954d4
User-Agent: PostmanRuntime/7.6.0
Accept: */*
Host: localhost:3000
cookie: csrfToken=58RWUaRa3ZuA2uIp7cxn34pC
accept-encoding: gzip, deflate
content-length: 157
Connection: keep-alive

----------------------------879095998142409176007484
Content-Disposition: form-data; name="x"

x
----------------------------879095998142409176007484--
```

net `http` `net`

```
var statusLine = `HTTP/1.1 ${statusCode} ${this.statusMessage}${CRLF}`; // line 252

function Server(options, requestListener) {
  net.Server.call(this, { allowHalfOpen: true });
  if (requestListener) {
    this.on('request', requestListener);
  }
} // line 283

net.Server.call(this, { allowHalfOpen: true }); //line 298
```

- Node
- tcp http

Revision #2
Created 20 March 2020 16:25:22 by
Updated 23 March 2020 10:03:24 by